

# Utilizing deep learning algorithms for the resolution of partial differential equations

Soumaya Nouna<sup>1,2</sup>, Assia Nouna<sup>2</sup>, Mohamed Mansouri<sup>2</sup>, Achhab Boujamaa<sup>2</sup>

<sup>1</sup>Department of Mathematics and Informatics, Hassan First University of Settat, ENSA Berrechid, Laboratory LAMSAD, Berrechid, Morocco

<sup>2</sup>Hassan First University of Settat, ENSA Berrechid, Laboratory LAMSAD, Berrechid, Morocco

## Article Info

### Article history:

Received May 29, 2024

Revised Jul 23, 2024

Accepted Aug 27, 2024

### Keywords:

Deep learning

Machine learning

Neural network

Partial differential equations

## ABSTRACT

Partial differential equations (PDEs) are mathematical equations that are used to model physical phenomena around us, such as fluid dynamics, electrodynamics, general relativity, electrostatics, and diffusion. However, solving these equations can be challenging due to the problem known as the dimensionality curse, which makes classical numerical methods less effective. To solve this problem, we propose a deep learning approach called deep Galerkin algorithm (DGA). This technique involves training a neural network to approximate a solution by satisfying the difference operator, boundary conditions and an initial condition. DGA alleviates the curse of dimensionality through deep learning, a meshless approach, residue-based loss minimisation and efficient use of data. We will test this approach for the transport equation, the wave equation, the Sine-Gordon equation and the Klein-Gordon equation.

*This is an open access article under the [CC BY-SA](#) license.*



## Corresponding Author:

Soumaya Nouna

Department of Mathematics and Informatics, Hassan First University of Settat

ENSA Berrechid, Laboratory LAMSAD

Berrechid, Morocco

Email: s.nouna@uhp.ac.ma

## 1. INTRODUCTION

Partial differential equations (PDEs) [1] can help us discover and understand the workings of nature, but most of these differential equations are impossible to solve due to their complexity and computationally intensive nature. For this reason, we use deep learning methods and exactly deep neural networks [2], [3] to solve mathematical problems. On the other hand, classical numerical methods [4] solve just one instance of the partial differential equation, unlike neural operators which study a complete group of partial differential equations, and also they immediately study the mapping of each function parameter dependent to the solution. This is why the field of artificial intelligence (AI) [5]-[7] is more important for a solution to these partial differential equations. Moreover, deep neural networks (DNNs) are able to provide solutions by solving problems without a specific amount of data. There are various different types of deep neural network architectures, however we will use long short-term memory (LSTM) network in our technique. The LSTM network, also known as LSTM, has been identified as the most successful recurrent neural network (RNN) [8] structure for the deep learning domain. The LSTM prevents the problem of the leakage gradient through the addition of the three gate structures: the forget gate, the entry gate, and the exit gate, by means of which the memory for the previous states may be effectively checked. The LSTM has been used extensively in various fields, primarily in machine learning (ML) applications domains.

Finite element technique [9], [10] approximates solutions by shape functions and Galerkin methods [11], [12] approximate solutions by basis functions. In contrast, deep Galerkin algorithm (DGM) use neural networks rather than basis functions and shape functions where these neural networks are capable of solving more complex systems. Our deep Galerkin algorithm (DGA) approach represents the natural fusion of Galerkin's approaches with ML [13], [14]. Also, DGM method may also be used to deal with first-order differential equations that are generally found in the field of finance [15]. The principal concept of the approach is to use deep neural networks to represent unknown functions. Notifying that where we reduce the losses associated with several operators and boundary conditions, a neural network can be trained. In addition, the neural network training data also consists of various possible function inputs generated through a random sample of the area over which a partial differential equation is being determined. One of the most distinctive properties of this method is that it is meshless, contrary to other numerical methods regularly used.

In this paper, we will apply a method that uses deep learning (DL) to solve partial differential equations. Specifically, this technique employs a deep neural network to approximate a solution of a PDE. Furthermore, stochastic gradient descent (SGD) has also been utilized for training the deep neural network at random sampled spatial points for satisfying the difference operator, the initial conditions, and the boundary conditions. The manuscript structure is presented in the following way: in this section 2, we present the theoretical part of LSTM, and the description of the deep neural network approach for solving the equations and their algorithm. Finally, in section 3 we provide some detailed calculation experiences to solve the PDEs, and in section 4 conclude.

## 2. METHOD (LSTM AND DGA)

Deep learning [16], [17] is simply a kind of ML, which is an inspiration for the human brain structure. DL techniques attempt to derive human-like conclusions by continuously examining data with a predefined logical framework. In order to succeed, DL utilizes a multi-layered architecture involving many algorithms known as neural networks. Moreover, neural network design focuses specifically on the human brain structure. Much like the way that we utilize the brain for identifying models or classifying various kinds of information, it is also possible to train neural networks to execute similar data processing tasks. Human brains behave in a similar way. Every time we acquire novel data, our brains try to associate the data with familiar items. This is a similar idea employed in deep learning.

In this section, we detail the architecture of the LSTM network used, the steps involved in formulating the method, and the training algorithm for solving the PDEs. We will also discuss some relevant theorems related to these networks. This innovative approach demonstrates how deep learning can be effectively applied to complex mathematical physics problems.

### 2.1. The LSTM structure

RNNs [18], [19] are used for persistent memory because it remembers preceding knowledge and is used to process any present input. However, due to the decreasing gradient, the RNN cannot remember long-term dependencies. Thus, to avoid the problems of long-term dependencies, we use the LSTM which is a more sophisticated RNN, a successive neural network that can retain knowledge. The LSTM [20]-[22] is working as an RNN cell. It contains three parts, which each serves a particular purpose. Part one is called Forget gate and selects if any information from the preceding timestamp should be memorized or if it is not relevant and may be discarded. The second part is known as an Input gate, where the cell attempts to obtain some novel information taken from entry into that cell. Finally, the third part is an output gate where a cell transmits upgraded information from the present time-stamp into the next time-stamp.

As a basic RNN, the LSTM has a hidden state, while  $H^{t-1}$  indicates the last time's hidden state, and  $H^t$  indicates the present time's hidden state. Furthermore, LSTM has a cell state described by  $C(t-1)$  and  $C(t)$  as the current and previous time-stamp correspondingly. Again, the hidden state is called short-term memory with the cellular state is called long-term memory (see Figure 1). Therefore, there are two sections to the LSTM equations. The input port  $I_t$ , forget port  $F_t$ , and output port  $O_t$  are all found in the first section. Cell state  $C_t$ , candidate cell state  $\tilde{C}_t$ , and final output  $H_t$  are included in the second section. The equations can be expressed mathematically as follows:

$$I_t = \sigma(W_I \cdot X_t + V_I \cdot H_{t-1} + b_I) \quad (1)$$

$$F_t = \sigma(W_F \cdot X_t + V_F \cdot H_{t-1} + b_F) \quad (2)$$

$$O_t = \sigma(W_O \cdot X_t + V_O \cdot H_{t-1} + b_O) \quad (3)$$

$$\tilde{C}_t = \tanh(W_C \cdot X_t + V_C \cdot H_{t-1} + b_C) \quad (4)$$

$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t \quad (5)$$

$$H_t = O_t \odot \tanh(C_t) \quad (6)$$

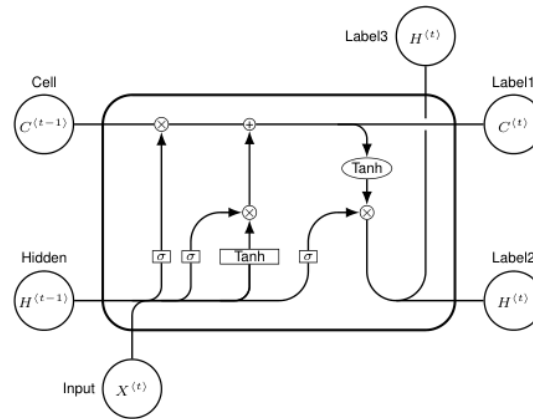


Figure 1. LSTM cell

$H_{t-1}$  is the output of the precedent LSTM phase (at  $t - 1$ ),  $X_t$  is the present timestamp's input,  $b_X$  represents the biases for the various ports,  $W_X$  represents the weight for the corresponding port neurons and  $\sigma$  represent an activation function. The most prominent advantages of LSTM neural networks are that the structure has the potential to successfully prevent leakage gradient phenomena and thus be chosen as an RNN Structure to identify the system for this document.

### 2.1.1. A neural network theorems

The following two important theorems for neural networks shall be introduced in this subsection: the theorem of Stone Weierstrass (Theorem 1) and the theorem of Universal Approximation (Theorem 2). In addition, with theorem 1, it is possible to show that the non-linear equations with some conditions are represented using the Wiener series. This leads to the discovery of the theorem of universal approximation.

Theorem 1, consider that  $\mathbf{X}$  is a compact space of Hausdorff and that  $\mathbf{B}$  is a sub-algebra at  $\mathbf{A}(\mathbf{X}, \mathbf{B})$  containing nonzero of constant features. So,  $\mathbf{B}$  can be dense into  $\mathbf{A}(\mathbf{X}, \mathbf{B})$  if only it can separate the points.

Theorem 2, lets  $\sigma$  be a continuous, non-constant, bounded, and monotonously increasing feature. Consider  $\mathbf{I}_n$  as an n-dimensional unitary hypercube  $[0; 1]^n$ . We denote the area of the continued functions onto  $\mathbf{I}_n$  with  $\mathbf{A}(\mathbf{I}_n)$ . So, for every function  $g \in \mathbf{A}(\mathbf{I}_n)$  and  $\xi > 0$ , it can exist a number integer  $M$ , some actual constants  $v_j, b_j \in \mathbb{R}$ , and some reals vectors  $w_j \in \mathbb{R}^n$ , with  $j = 1, \dots, M$ , so that we can determine:

$$\mathbf{G}(x) = \sum_{j=1}^M v_j \sigma(w_j^T x + b_j) \quad (7)$$

As the approximated solution for a function  $g$  in which  $g$  is independent of  $\sigma$ , i.e.,

$$|\mathbf{G}(x) - g(x)| < \xi \quad (8)$$

with every  $x \in \mathbf{I}_n$ . This means that the functions in form  $\mathbf{G}(x)$  are dense inside  $\mathbf{A}(\mathbf{I}_n)$ .

## 2.2. Description of methodology

In general, the type of nonlinear partial differential equations are defined in the following terms: let  $\mathbf{u}(\mathbf{t}, \mathbf{x})$  denote an unknowable function of time variable  $\mathbf{t}$  and space variable  $\mathbf{x}$  with  $d$  spatial dimensions. Let us suppose  $\mathbf{u}$  has the following partial differential equation:

$$\begin{cases} \partial_t \mathbf{u}(\mathbf{t}, \mathbf{x}) + \mathcal{L}\mathbf{u}(\mathbf{t}, \mathbf{x}) = 0, & \mathbf{t} \in [0, T], \mathbf{x} \in \Omega \in \mathbb{R}^d \\ \mathbf{u}(0, \mathbf{x}) = \mathbf{u}_0(\mathbf{x}), & \mathbf{x} \in \Omega \\ \mathbf{u}(\mathbf{t}, \mathbf{x}) = h(\mathbf{t}, \mathbf{x}), & \mathbf{t} \in [0, T], \mathbf{x} \in \partial\Omega. \end{cases} \quad (9)$$

With  $\partial\Omega$  is the limit of the  $\Omega$ -field and  $\mathcal{L}$  denotes a differential operator having the best properties. The objective of this approach consists in approximating  $\mathbf{u}(\mathbf{t}, \mathbf{x})$  using an approximate feature  $\mathbf{g}(\mathbf{t}, \mathbf{x}; \theta)$  generated with a neural network having a collection of parameters  $\theta$ . This training problem's loss function is composed of three components:

i) The measurement for the way that an approximation fulfills the operator of the differential:

$$\|\partial_t \mathbf{g}(\mathbf{t}, \mathbf{x}; \theta) + \mathcal{L}\mathbf{g}(\mathbf{t}, \mathbf{x}; \theta)\|_{[0, T] \times \Omega, \nu_1}^2 \quad (10)$$

ii) The measurement for the way that an approximation fulfills the boundary condition:

$$\|\mathbf{g}(\mathbf{t}, \mathbf{x}; \theta) - h(\mathbf{t}, \mathbf{x})\|_{[0, T] \times \partial\Omega, \nu_2}^2 \quad (11)$$

iii) The measurement for the way that an approximation fulfills the initial condition:

$$\|\mathbf{g}(0, \mathbf{x}; \theta) - \mathbf{u}_0(\mathbf{x})\|_{\Omega, \nu_3}^2 \quad (12)$$

The errors are expressed at items of  $L^2$ -norm in all three terms, i.e  $\|K(z)\|_{\mathbf{Z}, \nu}^2 = \int_{\mathbf{Z}} |K(z)|^2 \nu(z) dz$  with  $\nu(z)$  is the positive probability density on  $z \in \mathbf{Z}$ . By combining the above three elements, we obtain the loss function related to the training of the neural network:

$$J(\theta) = \|\partial_t \mathbf{g}(\mathbf{t}, \mathbf{x}; \theta) + \mathcal{L}\mathbf{g}(\mathbf{t}, \mathbf{x}; \theta)\|_{[0, T] \times \Omega, \nu_1}^2 + \|\mathbf{g}(\mathbf{t}, \mathbf{x}; \theta) - h(\mathbf{t}, \mathbf{x})\|_{[0, T] \times \partial\Omega, \nu_2}^2 + \|\mathbf{g}(0, \mathbf{x}; \theta) - \mathbf{u}_0(\mathbf{x})\|_{\Omega, \nu_3}^2 \quad (13)$$

the next step is using stochastic gradient descent (SGD) to optimize a loss function  $J$ . More specifically, we employ the following algorithm.

### 2.2.1. Algorithm

The deep Galerkin method (DGM) algorithm is described by Algorithm 1. It should be noted that the presented issue is essentially an optimization issue. To optimize the parameter  $\theta$  in this problem we can use the SGD algorithm [23] which takes averaged steps in a descending direction of function  $J$ , as in standard deep neural network training. We can also use the Adam optimizer [24] in our numerical results.

---

#### Algorithm 1

---

1. Initialize the learning rate  $\gamma_n$  and the parameter set  $\theta_0$ .
2. Generate random samples  $(\mathbf{t}_n, \mathbf{x}_n)$  based on  $[0, T] \times \Omega$  depending on  $\nu_1$  and  $(\tau_n, y_n)$  based on  $[0, T] \times \partial\Omega$  depending on  $\nu_2$  also  $\mathbf{w}_n$  based on  $\Omega$  depending on  $\nu_3$ .

3. Determine a loss function at points  $c_n = \{(\mathbf{t}_n, \mathbf{x}_n), (\tau_n, y_n), \mathbf{w}_n\}$ :

$$\text{Determine } J_1(\theta_n; \mathbf{t}_n, \mathbf{x}_n) = \left( \partial_t \mathbf{g}(\mathbf{t}_n, \mathbf{x}_n; \theta_n) + \mathcal{L}\mathbf{g}(\mathbf{t}_n, \mathbf{x}_n; \theta_n) \right)^2$$

$$\text{Determine } J_2(\theta_n; \tau_n, y_n) = \left( \mathbf{g}(\tau_n, y_n) - h(\tau_n, y_n) \right)^2$$

$$\text{Determine } J_3(\theta_n; \mathbf{w}_n) = \left( \mathbf{g}(0, \mathbf{w}_n) - \mathbf{u}_0(\mathbf{w}_n) \right)^2$$

$$\text{Determine } J(\theta_n, c_n) = J_1(\theta_n; \mathbf{t}_n, \mathbf{x}_n) + J_2(\theta_n; \tau_n, y_n) + J_3(\theta_n; \mathbf{w}_n)$$

4. At point  $c_n$ , consider a descent step:

$$\theta_{n+1} = \theta_n - \gamma_n \nabla_{\theta} J(\theta_n, c_n)$$

5. Replay (2) – (4) until  $\|\theta_{n+1} - \theta_n\|$  is little.
-

### 2.2.2. Implementing regulations

DGA network architecture is similar to that of LSTM. The deep Galerkin layers are composed of 3 layers, including the input, the hidden, and the output layer. Every deep Galerkin layer, on the other hand, receives as input an original small-batch input  $X$  (in our example, a group of randomly generated spatiotemporal elements) as well as the output of the preceding deep Galerkin layer. The output result of this process is a vector-valued  $Y$  that involves the neural network's approximation of the required function  $V$  estimated at the minibatch data points. In the DGA layer, the minibatch input and the preceding layer's output are converted via a sequence of actions. In (14) show the architecture.

$$\begin{aligned}
 C_1 &= \sigma(W_1 \cdot X + b_1) \\
 D_n &= \sigma(V_{d,n} \cdot X + W_{d,n} \cdot C_n + b_{d,n}) \quad n = 1, \dots, N \\
 K_n &= \sigma(V_{k,n} \cdot X + W_{k,n} \cdot C_n + b_{k,n}) \quad n = 1, \dots, N \\
 Q_n &= \sigma(V_{q,n} \cdot X + W_{q,n} \cdot C_n + b_{q,n}) \quad n = 1, \dots, N \\
 H_n &= \sigma(V_{h,n} \cdot X + W_{h,n} \cdot (C_n \odot Q_n) + b_{h,n}) \quad n = 1, \dots, N \\
 C_{n+1} &= (1 - K_n) \odot H_n + D_n \odot C_n \quad n = 1, \dots, N \\
 g(\mathbf{t}, \mathbf{x}; \theta) &= \mathbf{W} \cdot C_{N+1} + b
 \end{aligned} \tag{14}$$

With  $\odot$  represents Hadamard multiplication (element-by-element).  $N$  indicates a whole number of layers.  $\sigma$  denotes the activation function.  $b$ ,  $V$ , and  $\mathbf{W}$  represent features, while the different superscripts are parameters of the model. According to the LSTM concept, every layer generates weights depending on the previous layer to determine how often information is passed to the next layer.

## 3. NUMERICAL RESULTS AND DISCUSSION

Throughout this section, we use the deep Galerkin approach to solve several PDEs observed in the physical environment. Although previous studies have explored the application of neural networks to the numerical solution of PDEs, they have not explicitly addressed the many experimental and practical considerations necessary for successful implementation. This study examines these considerations in detail, including the design of the neural network, the balance between execution time and accuracy, the choice of activation functions and hyper-parameters, optimisation techniques, training intensity, and the programming environment.

We begin by stating a PDE with its exact solution, and then provide an approximate solution using the DGM. For all subsequent PDEs, we use the same network architecture introduced in Chapter 5 of [15], using Xavier initialization for the weights. The network has been trained over several iterations, which may vary between examples.

To generate the training datasets for the model, we used a uniformly distributed sampling method covering the function domain as well as the initial and terminal conditions. Points within the domain are generated by uniformly sampling time points  $t$  and space points  $x$  within the function domain. For boundary conditions, the time points are fixed at terminal time, and the space points are sampled uniformly over the same spatial interval.

The model training process follows the following steps:

1. Initialization: the neural network is initialized with the Xavier initialization for the weights, which helps maintain the gradient scale during back propagation.
2. Sampling: at each iteration, a new set of points is randomly sampled from the function domain for interior points and terminal conditions.
3. Residual calculation: the model calculates the residuals of the PDE, the boundary conditions and the initial conditions for the sampled points.
4. Loss minimisation: a loss function based on the residuals is minimised using optimisation techniques such as Adam. This loss function incorporates errors in the residuals, boundary conditions and initial conditions.
5. Weight update: the neural network weights are updated according to the gradients calculated from the loss function.
6. Repeat: this process is repeated for a defined number of iterations or until convergence is reached.

### 3.1. The transport equation

The transport equation can be known as the convection-diffusion equation, which describes how the scalar is transmitted in space. Generally, it is used for scalar field transport as material properties, temperature, or chemical concentration in incompressible flows. Here, the transport equation with the given initial condition is defined as (15).

$$\begin{cases} \partial_t u(t, x) + \partial_x u(t, x) = 0, & t \in [0, T], x \in \Omega \\ u(0, x) = \exp(-x^2), & x \in \Omega \end{cases} \quad (15)$$

The analytical solution of (15) is  $u_{ex}(t, x) = \exp(-(x - t)^2)$ , where  $\Omega = [0, 1]$  and  $T = 1$ . In this simulation, we utilize a three-layers neural network with fifty nodes per layer. We also sample uniformly in the temporal and spatial domains. Figure 2 shows a comparison between the solution by the DGM approach and the exact solution. The two solutions are almost identical, with a very low error (see Table 1). This high accuracy without a significant increase in computation time demonstrates the power and efficiency of the DGA method. Our results suggest that the DGA method is promising for future applications in solving PDEs.

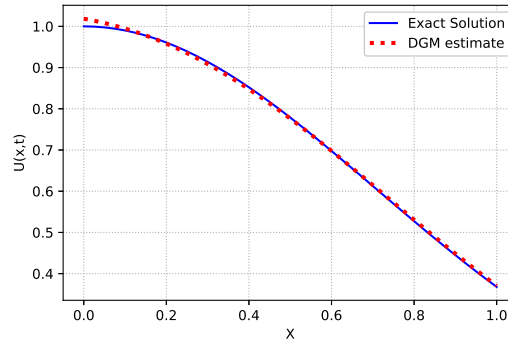


Figure 2. The transport equation: the deep Galerkin solution is shown in red, while the analytical solution is shown in blue. At  $t = 1$ , both solutions are confusing

### 3.2. The wave equation

The below equations are partial differential equations named wave equations, which can be used to simulate various phenomena, for example, vibrant strings and propagating waves. The dimension of the constant term  $v$  is  $m/s$ , which can be explained as wave velocity.

$$\begin{cases} \frac{\partial^2 u(t, x)}{\partial t^2} - v \frac{\partial^2 u(t, x)}{\partial x^2} = 0, & t > 0, x \in [-l, l] \\ u(0, x) = \frac{1}{2} \sin(x), & x \in [-l, l] \\ u(t, -l) = u(t, l), & t > 0. \end{cases} \quad (16)$$

$$\begin{cases} \frac{\partial^2 u(t, x)}{\partial t^2} - v \frac{\partial^2 u(t, x)}{\partial x^2} = 0, & t > 0, x \in [-l, l] \\ u(0, x) = \frac{1}{2} \cos(x), & x \in [-l, l] \\ u(t, -l) = u(t, l), & t > 0. \end{cases} \quad (17)$$

The exact solutions of (16) and (17) are  $u_{ex}(t, x) = \frac{1}{2}(\sin(x - vt) + \sin(x + vt))$  and  $u_{ex}(t, x) = \frac{1}{2}(\cos(x - vt) + \cos(x + vt))$  respectively, where  $l = \pi$  and  $v = 1$ . Consider that any function with parameters  $x - vt$  or  $x + vt$  a combination of both is a solution to the wave equation. This means we can simulate many different waves. Also, as you might have discovered, the exact solution is a combination of waves propagating to the left and waves propagating to the right.

We evaluate the DL algorithm on the wave equation. The methodology used to deal with boundary is to test consistently over the locale of intrigue and acknowledge/reject preparing models for that specific cluster of focuses, contingent upon whether or not they are inside or outside the limit district inferred by the last cycle of preparing. This methodology can efficiently retrieve the choice qualifications. As a result, we show that

the DGM approach precisely addresses the partial differential equations with a very small error (see Table 1). Figure 3 shows the comparison between the exact solution and the predicted solution for (16) and (17).

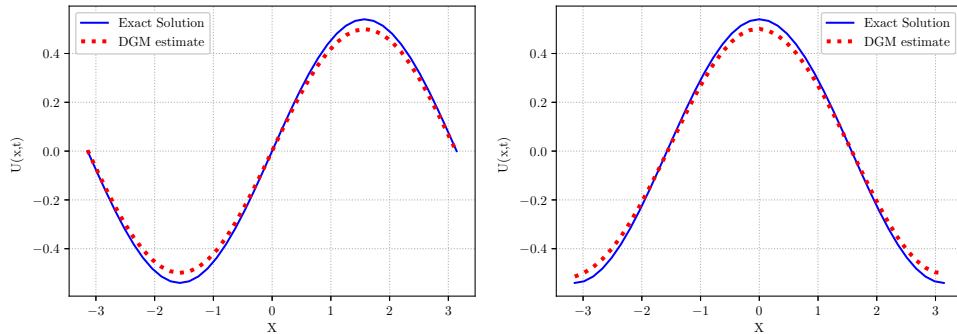


Figure 3. The wave equation: the deep Galerkin solution is shown in red, while the analytical solution is in blue. The figure on the left shows the simulation of (16), while the figure on the right shows the simulation of (17)

### 3.3. The Sine-Gordon equation

The Sine-Gordon equation is associated with the Korteweg de Vries and cubic Schrödinger equations, as all these equations recognize soliton solutions. This equation represents the nonlinear wave in the elastic medium. It is also used in many physical applications, such as in relativistic field theory and mechanical transmission lines. Can be seen in (18):

$$\begin{cases} \frac{\partial^2 u(t, x)}{\partial t^2} - \frac{\partial^2 u(t, x)}{\partial x^2} + \sin(u(t, x)) = 0, & t \in [0, T], x \in [0, L] \\ u(0, x) = 0, & x \in [0, L] \\ u_t(0, x) = 2\sqrt{2}\operatorname{sech}(\frac{x}{\sqrt{2}}), & x \in [0, L] \end{cases} \quad (18)$$

with  $L = 2\pi$  and the terminal time  $T = 6\pi$ . In (19) is the analytical solution named a breather soliton according to its oscillatory time evolution.

$$u_{ex}(t, x) = 4 \tan^{-1} \left( \frac{\sin(t/\sqrt{2})}{\cosh(x/\sqrt{2})} \right) \quad (19)$$

We also tested the DGM algorithm on the Sine-Gordon equation. The deep Galerkin solution is learned using a loss function to train all parameters of the 3-layer DGM network. Every hidden layer contains 50 hidden neurons. Our DGM should generally provide sufficient approximation capability to satisfy the complexities of  $u$ . Figure 4 shows the predicted and exact solutions of (18). Both solutions are confused with a small error (see Table 1), which shows that the proposed method has better accuracy.

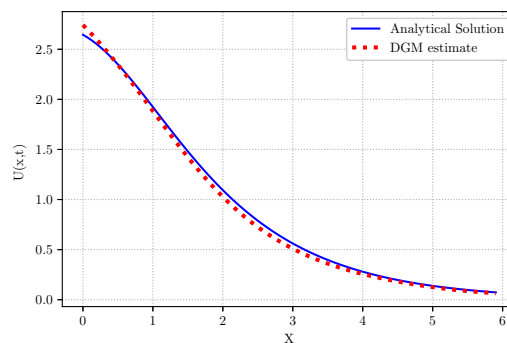


Figure 4. The Sine-Gordon equation: the comparison of predicted and exact solutions. Both solutions are confusing

### 3.4. The Klein-Gordon equation

The Klein-Gordon equation [25], also known as the Klein-Fock-Gordon equation or the Klein-Gordon-Fock equation, is an equation of relativity wave connected with Schroedinger's equation. Quanta are spin-less particles in this field, and the solutions to the Klein-Gordon equations consist of a pseudoscalar or quantum scalar field. The Klein-Gordon equation's fundamental theory is strongly linked to the Dirac equation. The Klein-Gordon standard can be seen in (20):

$$\begin{cases} \frac{\partial^2 u(t, x)}{\partial t^2} - \frac{\partial^2 u(t, x)}{\partial x^2} - u(t, x) = 0, & t \in [0, T], x \in [-L, L] \\ u(0, x) = 1 + \sin(x), & x \in [-L, L] \\ u_t(0, x) = 0, & x \in [-L, L] \end{cases} \quad (20)$$

with  $x \in [-2, 2]$  and  $t \in [0, 1]$ . The exact solution of this equation is defined as (21):

$$u_{ex}(t, x) = \sin(x) + \cosh(t) \quad (21)$$

The deep learning algorithm was also put to the test on the Klein-Gordon equation. The DGM is learned by training all of the parameters of the 3-layer DGM network using the loss function, with each hidden layer containing 50 hidden neurons. Figure 5 shows that the neural network model's predictions and exact solutions are coherent, demonstrating that the deep learning model can successfully solve the Klein-Gordon equation. Furthermore, the relative error for this example was calculated to be  $7,29.10^{-4}$  confirming the method's effectiveness. Despite the Klein-Gordon equation's enormous complexity, the deep learning model can produce results that are very near to the actual solution from the training data, demonstrating that the method has significant promise and utility and is worthy of further investigation.

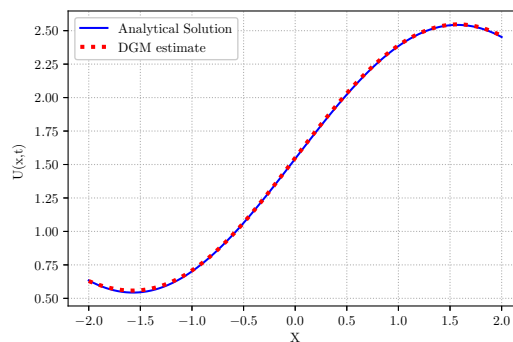


Figure 5. The Klein-Gordon equation: the exact solution and the predicted solution are confused

Table 1. The relative errors are obtained by the deep learning algorithm for each equation

PDEs	Deep Galerkin method				
	Equation (15)	Equation (16)	Equation (17)	Equation (18)	Equation (19)
Error	$1,87.10^{-4}$	$6,7.10^{-4}$	$1,8.10^{-4}$	$1,24.10^{-4}$	$7,29.10^{-4}$

We applied the DGA to solve various PDEs and compared the solutions obtained with the exact solutions. The results show a strong correlation between the DGA solutions and the exact solutions, with high accuracy without a significant increase in computation time. Compared with other methods, DGA offers improved accuracy and remarkable efficiency. Although our study demonstrated the viability of DGA, further research is needed to confirm its robustness for more complex PDEs. Our results suggest that the DGA is promising for future applications, and optimisation of the hyper-parameters could further improve its performance.







#### 4. CONCLUSION

We are confident that deep learning can serve as a beneficial approach to solving partial differential equations. This study presents a training methodology that leverages the inherent capabilities of neural networks for approximating solutions to partial differential equations. The proposed method employs deep neural networks to represent unknown functions that satisfy a given partial differential equation and form a network while minimizing the loss function associated with this problem. Moreover, instead of forming a mesh, the method employs a neural network that has been trained using batches consisting of random temporal and spatial data points. The transport and wave equations are used to demonstrate the effectiveness of the method, with accurate results obtained. Furthermore, the precision of the approach is evaluated on Sine-Gordon and Klein-Gordon equations, with computational findings demonstrating the approach's ability to attain good performance in terms of precision and prediction robustness. These findings provide sufficient evidence to warrant further research into deep learning methods for solving partial differential equations.





#### REFERENCES

- [1] P. K. Jakobsen, "An introduction to partial differential equations," *arXiv preprint*, 2019.
- [2] A. L. Caterini and D. E. Chang, *Deep neural networks in a mathematical framework*. Cham: Springer International Publishing, 2018.
- [3] Z. Deng, J. Shi, and J. Zhu, "NeuralEF: deconstructing Kernels by deep neural networks," *Proceedings of Machine Learning Research*, vol. 162, pp. 4976–4992, 2022.
- [4] Y. Wang, Y. Yan, and Y. Hu, "Numerical methods for solving space fractional partial differential equations using hadamard finite-part integral approach," *Communications on Applied Mathematics and Computation*, vol. 1, no. 4, pp. 505–523, 2019, doi: 10.1007/s42967-019-00036-7.
- [5] S. Dick, "Artificial intelligence," *Harvard Data Science Review*, Jun. 2019, doi: 10.1162/99608f92.92fe150c.
- [6] M. Mirbabaie, F. Brünker, N. R. J. M. Frick, and S. Stieglitz, "The rise of artificial intelligence – understanding the AI identity threat at the workplace," *Electronic Markets*, vol. 32, no. 1, pp. 73–99, Mar. 2022, doi: 10.1007/s12525-021-00496-x.
- [7] D. Minh, H. X. Wang, Y. F. Li, and T. N. Nguyen, "Explainable artificial intelligence: a comprehensive review," *Artificial Intelligence Review*, vol. 55, no. 5, pp. 3503–3568, Jun. 2022, doi: 10.1007/s10462-021-10088-y.
- [8] F. M. Talaat, "Explainable enhanced recurrent neural network for lie detection using voice stress analysis," *Multimedia Tools and Applications*, vol. 83, no. 11, pp. 32277–32299, Sep. 2024, doi: 10.1007/s11042-023-16769-w.
- [9] P. Gervasio and A. Quarteroni, "The INTERNODES method for non-conforming discretizations of PDEs," *Communications on Applied Mathematics and Computation*, vol. 1, no. 3, pp. 361–401, Sep. 2019, doi: 10.1007/s42967-019-00020-1.
- [10] J. J. Connor and C. A. Brebbia, *The Finite Element Technique*, 1976.
- [11] Y. Liu, C. W. Shu, and M. Zhang, "Superconvergence of energy-conserving discontinuous Galerkin methods for linear hyperbolic equations," *Communications on Applied Mathematics and Computation*, vol. 1, no. 1, pp. 101–116, 2019, doi: 10.1007/s42967-019-0006-y.
- [12] J. Badwaik, P. Chandrashekar, and C. Klingenberg, "Single-step arbitrary lagrangian–eulerian discontinuous Galerkin method for 1-D euler equations," *Communications on Applied Mathematics and Computation*, vol. 2, no. 4, pp. 541–579, 2020, doi: 10.1007/s42967-019-00054-5.
- [13] I. El Naqa and M. J. Murphy, "What is machine learning?," in *Springer International Publishing*, 2015, pp. 3–11.
- [14] P. P. Shinde and S. Shah, "A review of machine learning and deep learning applications," in *Proceedings - 2018 4th International Conference on Computing, Communication Control and Automation, ICCUBEA 2018*, Aug. 2018, pp. 1–6, doi: 10.1109/IC-CUBEA.2018.8697857.
- [15] J. Sirignano and K. Spiliopoulos, "DGM: a deep learning algorithm for solving partial differential equations," *Journal of Computational Physics*, vol. 375, pp. 1339–1364, Dec. 2018, doi: 10.1016/j.jcp.2018.08.029.
- [16] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, doi: 10.1038/nature14539.
- [17] T. T. Khoei, H. O. Slimane, and N. Kaabouch, "Deep learning: systematic review, models, challenges, and research directions," *Neural Computing and Applications*, vol. 35, no. 31, pp. 23103–23124, Nov. 2023, doi: 10.1007/s00521-023-08957-4.
- [18] A. Sherstinsky, "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network," *Physica D: Nonlinear Phenomena*, vol. 404, p. 132306, Mar. 2020, doi: 10.1016/j.physd.2019.132306.
- [19] J. Bohn and M. Feischl, "Recurrent neural networks as optimal mesh refinement strategies," *Computers and Mathematics with Applications*, vol. 97, pp. 61–76, Sep. 2021, doi: 10.1016/j.camwa.2021.05.018.
- [20] B. Zraibi, M. Mansouri, and C. Okar, "Comparing single and hybrid methods of deep learning for remaining useful life prediction of lithium-ion batteries," *E3S Web of Conferences*, vol. 297, p. 01043, Sep. 2021, doi: 10.1051/e3sconf/202129701043.
- [21] R. C. Staudemeyer and E. R. Morris, "Understanding LSTM – a tutorial into long short-term memory recurrent neural networks," *arXiv preprint*, 2019, [Online]. Available: <http://arxiv.org/abs/1909.09586>.
- [22] R. Wang, C. Peng, J. Gao, Z. Gao, and H. Jiang, "A dilated convolution network-based LSTM model for multi-step prediction of chaotic time series," *Computational and Applied Mathematics*, vol. 39, no. 1, p. 30, Mar. 2020, doi: 10.1007/s40314-019-1006-2.
- [23] L. Bottou, "Stochastic gradient descent tricks," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pp. 421–436, 2012, doi: 10.1007/978-3-642-35289-8\_25.
- [24] J. L. Ba and D. P. Kingma, "Adam: a method for stochastic optimization," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–15, 2015.
- [25] M. Mirzazadeh, M. Eslami, and A. Biswas, "Soliton solutions of the generalized Klein–Gordon equation by using (formula presented)-expansion method," *Computational and Applied Mathematics*, vol. 33, no. 3, pp. 831–839, 2014, doi: 10.1007/s40314-013-0098-3.





**BIOGRAPHIES OF AUTHORS**

**Soumaya Nouna**     is a researcher at the systems analysis and modelling and decision support research laboratory at Hassan First University in Settat. She is an expert in mathematics, ML and DL. A doctoral researcher in mathematics and computer science, she brings a wealth of experience to her field. Her skills include the analysis of differential equations, and ML algorithms. Soumaya Nouna is also the author of numerous research articles and is constantly seeking to advance in her areas of expertise. She can be contacted at email: s.nouna@uhp.ac.ma.







**Assia Nouna**     is a researcher at the systems analysis and modelling and decision support research laboratory at Hassan First University in Settat. A doctoral researcher in mathematics and computer science. She is currently working on deep learning and satellite imagery for agricultural applications. Her research aims to enhance agricultural practices through precise soil analysis, improving crop management and yield predictions. Additionally, she has contributed to various projects and publications in the field, demonstrating her expertise in applying advanced computational techniques to solve real-world problems. She can be contacted at email: a.nouna@uhp.ac.ma.



**Mohamed Mansouri**     received the Ph.D. degree in Mechanical Engineering and Engineering Sciences from the faculty of science and technology, Hassan First University, Settat, Morocco, and from L'INSA, Rouen, France, in 2013. He is currently a Professor and researcher at the National School of Applied Sciences in Berrechid, Department of Electrical Engineering and Renewable Energies. His research interests include Mechano-reliability study, industrial engineering, optimization of shape and reliability optimization of coupled fluid-structure systems, and energy storage systems. He can be contacted at email: m.mansouri@uhp.ac.ma.



**Achhab Boujamaa**     is a professor and director at ENSA Berrechid, Hassan 1st University, specializing in applied mathematics and computer science. He completed his Ph.D. at Université Claude Bernard Lyon 1 in 1995. His research focuses on numerical analysis, mathematical modeling, and computational finance. Notable works include simulations of the Black-Scholes equation and studies on stochastic processes. Achhab is proficient in various mathematical and simulation software, with strong analytical skills and experience in collaborative research projects. He can be contacted at email: achhab@yahoo.fr.